**Defect Analysis**

Analyses of discovered defects and related information from quality assurance (QA) activities can help both developers and testers to detect and remove potential defects, and help other project personnel to improve the development process, to prevent injection of similar defects and to manage risk better by planning early for product support and services. The defect data are typically collected from the main QA activities. Some additional details regarding the defects may need to be collected during this process or extracted from some system records to provide better quality assessments, predictions, or identification of problematic areas. We next discuss these topics, and illustrate them through several case studies analyzing defects from system testing for some IBM products, and web-related defects for www . seas. smu. edu, the official web site for the School of Engineering and Applied Science, Southern Methodist University (SMUKEAS).

**Questions Asked During The Analyses**

**What?** The identification and classification of the discovered defects can be performed to identify what they are and classify them by some consistent scheme. This topic is the focus of this chapter, which is discussed in all subsequent sections.

**Where?** Where was the defect found or discovered? This information can be used to provide valuable feedback to the development process through defect distribution analysis.

**When?** The identification of the exact time or associated development phase or subphase when a defect is injected and when it is discovered is important, because it provides information to analyze the overall defect trend and serves as the basis for quality prediction into the future. This topic is discussed later in this section, and finegrain defect timing analysis is covered in Chapter 22 as part of software reliability engineering.

**Pre- or post-release?** An important extension to the “when” question is whether a defect is a pre-release defect or a post-release defect, sometimes labeled as an indevelopment (or in-process) or an in-field defect, respectively. Although the in-field defects are the ones experienced by actual customers or users, and should receive adequate attention, the scarcity of post-release data and business-sensitive information they might contain leave most existing software engineering research with the use of pre-release defect data only. This issue is discussed in relation to specific topics throughout this chapter.

**How and Why?** How was the defect injected into the software, and why? These two questions are closely related, both pertaining to the cause of the discovered defects. Notice that all the analyses listed above are applied to defect information as the primary target or focus. However, other information and measurements related to defect information are often needed in these analyses, although sometimes used implicitly.

**Defect Distribution Analysis**

Defect distribution analyses can help us answer the what and where questions above. In answering the what question, we can find out the distribution of defects over different defect types, and if certain defect types are associated with an overwhelming share of the overall defects. If the latter is confirmed to be true, the identification of these dominant defect types can help us select appropriate remed.ia1 or corrective actions to effectively address the problems and improve product quality. Similarly, in answering the where question, we can find out the distribution of defects over different areas or product components, and if there are certain areas that are associated with an overwhelming share of the overall defects. If the latter is confirmed to be true, the identification of these high-defect areas can help us focus our remedial or corrective actions to effectively improve product quality. The defect distribution analyses typically deal with faults or defect fixes instead of failures or errors we defined in Chapter 2. “Defect fixes” is typically used if actual fixing of discovered problem took place before defect analyses were performed, while “faults” can be used as long as it is identified (but not necessarily fixed already). Defect fixes, labeled DF in this book, are in response to observed failures during testing or to discoveries of other problems during development or operation. We selected defect fixes instead of raw defect counts because much of the defect propagation information is captured in the former but not in the latter. Defect propagation is affected by the system structure, the interconnection among different components, and product evolution. DF can be identified with specific modules, therefore permitting analysis and modeling using various software metrics defined on modules. Both the pre-release and post-release defect data can be analyzed and compared.

For more information see the specified book plz…

**Defect Trend Analysis**

Most of the defect data contains some timing information. At a minimum, the discovered defect is classified as either pre-release or post-release. This information can be used to give us a general picture of the defect trend. When used with appropriate models, these data can provide us with the basis for prediction into the future. Sometimes, timing information for individual defects corresponds to some rough information about the development phases or sub-phases recorded in relevant defect records. When such information is available, we can examine the defect distribution over these phases or sub-phases, much like the distribution analysis described above, but with phases or sub-phases along some timeline. The defect removal model in the previous chapter can be considered an example of such a trend analysis. If the information about defect injection time is available, it can be used to augment the defect removal models into the so called defect dynamics model, where both the injection and removal of defects are tracked by development phases. This model is often represented as a matrix, such as in Table 20.5, with the rows corresponding to defect injections in each phase, and column corresponding to defect removals in each phase. The inner matrix is always an upper triangular matrix because the removal of a defect is always after its injection. The last row, summing up all the defects removed in different phases, actually gives us a defect removal model similar to the one given in the previous chapter. The last column, summing up all the defects injected in different phases, gives us information about where the major defect sources are in terms of when they are injected. However, the cost of each defect injected in phase X and removed in phase Y is not uniform. Typically, the cost increases substantially with the increase of the distance between X and Y, or the number of phases when a defect lies dormant. Because a dormant defect might trigger the injection of other related defects, and the further away a defect is removed from when it is injected, the harder it gets to remove it because of all the intermediate decisions and actions applied that obscure the linkage between causes and effects. Consequently, the focus of defect dynamics models is typically on the off-diagonal ones, or those out-of-phase defect removals. In addition, when the post-release defect data are available, they deserve more attention as well, because these defects are the ones that escaped the software QA process to cause real damage to the customers and users. They also harm the development organizations’ reputation and may lead to product liability problems. When precise time information about the defect discoveries is available, it can be used in various models of greater precision to provide finer-grain or better quality predictions. For example, the Putnam model (Putnam, 1978) described in the previous chapter is an example of such a model. Various software reliability growth models (SRGMs) to be described in Chapter 22 can also be considered examples of fine-grain defect trend models. However, typically other measurement data, such as testing or usage activities, are needed for analyses with SRGMs. On the other hand, precise defect injection time information is typically impossible to obtain, depriving us of the fine-grain defect injection or defect dynamics models.

**Defect Causal Analysis**

Defect causal analysis can usually take two forms: logical analysis and statistical analysis. Logical analysis is a deterministic analysis that examines the logical link between the effects and the corresponding causes, and establishes general causal relations. Statistical analysis is a probabilistic analysis that examines the statistical link between causes and effects and deduces the probable causal relations between the two. The effects in the defect causal analysis can be either the observed failures or discovered (or fixed) faults, and the corresponding causes are the faults that caused the failures or the errors that caused the injection of the faults, respectively. The causal relations between faults and failures typically are determined by the developers or code owners who fix the code or design in response to failure observations during testing, inspection, or normal operational usage, as part of the normal development process where defects are fixed. The causal relations between errors and faults are typically determined through dedicated defect causal analysis beyond the normal development process. This kind of defect causal analysis, particularly its logical instead of the statistical variation, is also referred to as root cause analysis in literature. Root cause analysis is human intensive, and should be performed by experts with thorough knowledge about the product, the development process, the application domain, and the general environment. Sometimes, it can be integrated into the development or specific QA process. For example, in the Gilb inspection (Gilb and Graham, 1993) described in Chapter 14, a phase called process brainstorming is added between inspection meetings and follow-up actions. This process brainstorming is essentially a root cause analysis. Some times, root cause analysis can be performed selectively, for example, for all the critical defects. Statistical analysis is based on empirical evidence collected either locally or from other similar projects. These empirical data can be fed to various models to establish the predictive relations between causes and effects. Once such causal relations are established, appropriate QA activities can then be selected and applied for fault or error removal. This kind of analyses employ various statistical models. For example, the simplest of such models is correlation analysis, which is often performed between defects and product internal measurements. For example, we may find that for a product, the number of defects per module may be closely correlated to module control flow complexity. Then we can conclude that high control flow complexity is probably the cause for the modules to have high defect, and focus our attention on the high-complexity modules in our QA activities even before defects are discovered. This risk focus, or focus on high-risk or potentially high-defect areas or product components, is the primary usage of statistical defect causal analysis